Software Reliability Corroboration

Carol Smidts**, Bojan Cukic, Erdogan Gunel*, Ming Li**, Harshinder Singh*, Lan Guo

Lane Department of Computer Science and Electrical Engineering
* Department of Statistics
West Virginia University
Morgantown, WV 26506-6109
E-mail: {cukic,lan}@csee.wvu.edu, {egunel, hsingh}@stat.wvu.edu

** Reliability Engineering Program
Department of Materials and Nuclear Engineering
University of Maryland
College Park, MD 20742
E-mail: {csmidts,mli}@eng.umd.edu

1. Introduction

Software reliability is a quantitative measure of software quality. It is defined as a probability of failure free execution given a specific environment and a fixed time interval. The goal of software reliability assessment is not just to estimate the failure probability of the program, θ, but to gain statistical confidence that θ is realistic. In practice, the required failure probability θ_0 and the confidence level C are application specific and predefined.

The input domain reliability assessment approach defines the reliability of a program as the probability of failure free operation for specific inputs [3]. Program P is seen as a function that maps all the elements of the multidimensional input space I into the output space O. The inputs that activate faults in the program are mapped into failures, i.e., incorrect outputs.

The traditional theory behind input domain models assumes that the certification testing process starts with "zero knowledge" about the system under test. In other words, all the observations collected during the development and verification and validation activities, such as inspections and reviews, module testing, formal analysis, engineering observations and judgments, etc., play no role in reliability certification. Consequently, the most important drawback of the input domain models is an enormous amount of testing (statistically independent executions of test cases) needed to attain a modest confidence that moderate software reliability has been achieved.

Process improvements and adherence to maturity models enable the production of software with repeatable quality [7, 9]. Ignoring process and product quality measurements collected throughout the development lifecycle in the software reliability certification phase is a mistake. On the other hand, reliability prediction models based on process and product measurements alone may not be sufficiently accurate [8, 15]. These predictions need *corroboration*. If during the certification testing we assume a limited belief in the accuracy of early assessment models then the main drawback of input domain reliability assessment models, the impractically large number of statistical tests, disappears. In other words, it is faster to confirm an existing belief (that the program is reliable enough) than to establish this belief by assuming nothing. In this paper, we describe the statistical theory behind software reliability corroboration. We believe that this approach has the long term potential to make software certification of high assurance systems practical**.**

The rest of the paper is organized as follows. Section 2 overviews traditional input domain based software reliability assessment models and their limitations. In Section 3, Bayesian hypothesis testing is presented. Section 4 describes an application of the theory to a small control software. The priors used are borrowed from a study on software engineering measures and their relationship to reliability. Section 5 summarizes our findings and provides directions for future work.

2. Input Domain Based Models: Background and Limitations

An intuitive measure of software reliability is the proportion of test cases that result in correct outputs. If n represents the total number of test cases and n_f the number of detected failures, the estimated reliability according to the Nelson model [16], is

$$\hat{R} = \frac{n - n_f}{n} = 1 - \frac{n_f}{n}.$$

When an indefinite number of test runs is taken,

$$\hat{R} = 1 - \lim_{n \to \infty} \frac{n_f}{n},$$

the fraction n_f/n in its limit represents the estimated probability of failure in a single run of the program. This leads to the reliability prediction based on the probability of correct execution in each run. Thus, the probability of correct execution over *i* runs is given by

$$\hat{R}(i) = (\hat{R})^i.$$

A sound foundation for reliability assessment in the input domain is provided by statistical sampling theory [3]. A program under test separates the input domain into two disjoint classes: inputs that are correctly and those that are incorrectly mapped into the corresponding outputs. Testing can be simulated by randomly drawing balls from an urn containing black balls (test cases in the input space *I* resulting in a failed execution) and white balls (correct executions). Consequently, from the statistical viewpoint, each test case is a Bernoulli trial, and the model is frequently called the *sampling model*. Since input domains are large, the likelihood of selecting the same test case *i* is small. Thus, reliability estimation assumes *sampling with replacement* in which nothing precludes repetition of a test case, but it is simpler (and cheaper) to implement. Due to the use of statistical sampling, authors often introduce statistical terminology [3, 17]. For example, the set of points in the input space is called the *population*, while the number of executed test cases during program testing is called the *sample size*.

Reliability assessment of software systems based on statistical sampling is performed in the certification phase of the software life-cycle. Faults are not removed when discovered. Rather, in the extreme case when ultra-high reliability is required, the program is rejected. Eventual assessment of the corrected program must be restarted from the beginning [13]. Often, it is unlikely that the program will fail during the certification testing, possibly due to the use of formal methods, fault prevention, and other techniques. Research has been performed concerning the difficult problem of estimating the probability of failure when a program does not fail. Classic statistical work dating back to Laplace states that when *t* white balls and no black balls are drawn from the urn containing an unknown proportion of black and white balls, the probability of drawing a black ball next (representing the probability of failure in a single run of the program) is $\frac{1}{t+2}$. The above result is known as Laplace rule of succession [5]. It means that failure-free testing relates the estimated probability of failure to the number of test cases. In order to establish the probability of failure at less than, say, 10^{-9} failures per hour, one needs to test the program for 10^9 hours (approximately *114,000* years) [4]. Limited improvement is achievable through the acceleration of testing. The second possibility for improvement is making prior assumptions about the failure probability [6], as presented in later sections. The central question is how much testing should be conducted?

3. Bayesian Hypothesis Testing

Let $0<\theta_o<1$ be a sufficiently small number close to zero that represents the required system reliability and let *C* represent the confidence level. Let us consider the null hypothesis H_o: $\theta \leq \theta_o$ and the alternative hypothesis H_1: $\theta > \theta_o$. The null hypothesis states that the program's true reliability (which is unknown, i.e., being estimated) is higher than required, whereas H_1 states the opposite, i.e., the system should not be released. In classical statistics a statistical hypothesis testing procedure is evaluated in terms of the Type I and Type II error probabilities. Type I error occurs when H_o is rejected when it is true and Type II error occurs when H_o is accepted when it is not true. In Bayesian analysis the task of deciding between H_o and H_1 is conceptually more appealing and, at the same time, more straightforward. We simply compute

$$P(\theta \leq \theta_o \mid \textit{Test data}),$$

the so called posterior probability of the null hypothesis H_o. The conceptual advantage is that the posterior probability reflects the prior opinions (the opinions about θ prior to certification testing of the program) and the results of the actual certification test. Let $P(H_o)$ and $P(H_1)$, where $P(H_o)+P(H_1)=1$, denote the prior probabilities assigned to the null and the alternative hypothesis. In practice, it may be difficult to obtain these probabilities, as discussed in the next section. Then,

$$O(H_o) = P(H_o) / P(H_1)$$

is called the *prior odds of H_o to H_1* and

$$O(H_o \mid \textit{Test Data}) = P(H_o \mid \textit{Test Data}) / P(H_1 \mid \textit{Test Data})$$

is called the *posterior odds ratio of H_o to H_1*. The *Bayes factor* $F(H_o, H_1)$ is defined as the ratio of

posterior odds to prior odds in favor of the null hypothesis,

$$F(H_o, H_1) = O(H_o \mid Test\ Data) / O(H_o) .$$

If the Bayes factor $F(H_o, H_1)$ is greater than one then we have evidence in favor of the null hypothesis and if it is less than one we have evidence against the null hypothesis. If the Bayes factor is equal to one then we do not discriminate between the null and the alternative hypothesis. The posterior probability of H_o can be written in terms of the prior probability of H_o and the Bayes factor,

$$P(H_o|Test\ Data) = P(H_o)F(H_o, H_1) / [P(H_o)F(H_o, H_1)+(1-P(H_o)] .$$

During certification testing, program executions either result in a success or in a failure. An important factor determining our ability to corroborate the null hypothesis is the number of failures in n tests of the program. We are interested in finding the overall number of certification tests, given the number of failures observed in certification, such that

$$P(H_o \mid Test\ Data) = C,$$

where C represents the required confidence level.

In Table 1, we give the number of required certification tests, assuming $C=0.99$, when no failures are encountered (column n_0), one failure is encountered (column n_1) and when two failures are encountered (column n_2), for some selected values of θ_o and $P(H_o)$.

A note of caution is appropriate here. This section presents a simple overview of the reasoning and justification behind the Bayesian hypothesis testing approach to determine the number of "software reliability corroboration tests". In theory, the prior beliefs in the null hypothesis and its alternative need to be provided as probability distributions of software failures over intervals (θ_o, 1) and (0, θ_o), respectively. In Table 1, our assumption is that the distribution of θ under H_o and H_1 is uniform. This by no means represents model limitation, since the theory behind this framework allows for any distribution.

Table 1. The number of tests required for reliability corroboration according to Bayesian hypothesis testing theory.

θ_o	$P(H_o)$	n_0	n_1	n_2
0.01	0.01	457	476	497
0.001	0.01	2378	2671	2975
0.0001	0.01	6831	10648	14501
0.00001	0.01	9349	33176	63649
0.000001	0.01	9752	101273	282007
0.01	0.02	388	410	433
0.001	0.02	1766	2098	2438
0.0001	0.02	3954	7549	11315
0.00001	0.02	4736	23037	49499
0.000001	0.02	4838	70800	221022
0.01	0.1	228	258	289
0.001	0.1	636	1017	1402
0.0001	0.1	853	3157	6150
0.00001	0.1	886	9646	27281
0.000001	0.1	890	30067	123725
0.01	0.4	90	128	167
0.001	0.4	138	411	739
0.0001	0.4	146	1251	3260
0.00001	0.4	147	3889	14724
0.000001	0.4	147	12222	67468
0.01	0.6	50	87	126
0.001	0.6	63	269	552
0.0001	0.6	65	827	2458
0.00001	0.6	65	2584	11173
0.000001	0.6	65	8139	51351

A closer look at the values in Table 1 reveals the strongest possible motivation for the software reliability corroboration approach. For $\theta_o=10^{-2}$ and $P(H_0)=0.4$, for example, if after *90* tests there is no failure then we are *99%* confident that $\theta \leq 10^{-2}$. If after *128* tests there has been one failure observed then we are *99%* confident that $\theta \leq 10^{-2}$. If after *167* tests there are *2* failures then we are, again, *99%* confident that $\theta \leq 10^{-2}$.

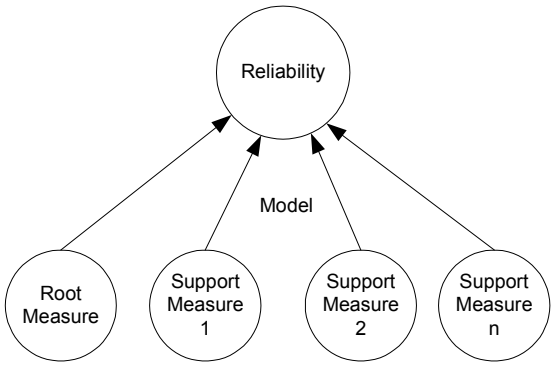
4. Predicting Software Reliability Prior to Certification

4.1. Priors Based on Software Engineering Measures

To be able to apply the framework discussed in Section 3 acceptable priors need to be identified and or developed. A possible source of priors is the set of priors that can be developed from software engineering measures such as fault density, cyclomatic complexity, requirements traceability, etc. In a study carried out for the U.S. Nuclear Regulatory Commission, 40 software engineering measures were ranked with respect to their ability to predict reliability [11, 14] for different phases of the life-cycle.

An initial validation of this ranking was performed in a follow-up study [10]. The study involved the following software engineering measures: requirements traceability, function points, bugs per line of code (Gaffney estimate), fault density and test coverage. To validate the ranking estimations of reliability were built based on these measurements. The validation study was limited to the testing phase and the estimates were used to predict reliability in operation. The application studied was a small control software, PACS. PACS is a simplified version of an automated personnel entry access system (gate) used to provide privileged physical access to rooms/buildings [1].

Figure 1. Reliability Prediction System (RPS).



The estimates were built using the concept of Reliability Prediction System (RPS). A RPS is a complete set of measures by which reliability can be predicted. The RPS is composed of a root measure and several support measures as shown in Figure 1. In Table 2 we provide the RPS for the measure "Test Coverage" as well as the model used to transform the measurements (the root and support measures) into a reliability prediction. Table 3 gives predictions for θ derived from the 5 measures considered in the validation study.

Table 2. RPS for "Test Coverage"

Measure	***Model***
Test coverage	$p_s = e^{-\frac{K\tau}{T_L}\times\frac{N^0}{a_0 \ln(1+a_1(e^{a_2\frac{LOC_T+k\times FP_M}{LOC_I+k\times FP_M}}-1))}}$

RPS	Notation	
Root measure: test coverage	C^0	defect coverage
Support measure:	C_1	test coverage (statement coverage)
• Implemented LOC (LOC_I)	a_0, a_1, a_2	coefficients
• Tested LOC (LOC_T)	N^0	the number of defects found by test
• The number of defects found by test (N^0)	N	the number of defects remaining
• Missing function point (FP_M)	K	fault exposure ratio
• Backfiring coefficient (k)	T_L	linear execution time
• Defects found by test (D_T)	τ	the average execution time per demand
• Linear execution time (T_L)		
• Execution time per demand (τ)		
• Fault exposure ratio (K)		

Table 3. Predicted θ Values

Measure	θ
Defect density	0.078
Test coverage	0.092
Requirements traceability	0.078
Function point	0.0020
Bugs per line of code (Gaffney estimate)	0.000028

4.2. From Priors to $P(H_0)$

The next step is to derive a value for $P(H_0)$ from the prior estimates given in Table 3. Let us assume for the purpose of this paper that θ_0 is equal to $9\text{x}10^{-2}$. Each of the probability estimates in Table 3 is the solution of a model based on a different RPS. This model, say the ith model M_i is a function of given measurements $m(M_i)$ and parameters $\Lambda(M_i)$. Consider for instance the model built for "Test Coverage". This model is a function of the measures $m(M_i)$={Implemented LOC, Tested LOC, Missing Function Point, Backfiring Coefficient, Defects Found by Test, Linear Execution Time, Execution Time Per Demand, Fault Exposure Ratio} and the parameters $\Lambda(M_i)=\{a_0, a_1, a_2\}$.

The estimate of interest is $U(\theta_0-\theta)$ (i.e. whether or not the null hypothesis is satisfied) where U stands for the step function: $U(x-x_0) = 1$ for $x \geq x_0$ and 0 otherwise.

The expected value of the unknown $U(\theta_0-\theta)$, $E(U(\theta_0-\theta))$ is thus given by:

$$E(U(\theta_0-\theta)) = P(H_0) = \sum_{i=1}^{n}[\iint U(\theta_0-\theta(M_i,m(M_i),\phi(M_i))dm(M_i)d\phi(M_i)]p(M_i)$$

where

$\theta(M_i, m(M_i), \phi(M_i))$ is the probability of failure estimate, solution of model M_i given measurements $m(M_i)$ and model parameters $\phi(M_i)$ and $p(M_i)$ is the probability that model M_i is correct. The concept of a correct model is not a new concept. Indeed it is borrowed from the model uncertainty literature. For more details on this topic the reader is referred to [2, 12].

In this paper the possible contributions of parameters' and measurements' uncertainty are neglected. Future research will investigate their impact in detail.

On the other hand, estimates of $p(Mi)$ can be obtained readily from the expert opinion elicitation process performed in the NRC study. Indeed, one of the criteria considered during the process is the "Relevance To Reliability" criterion. This criterion assesses the perceived "distance" between a root measure and reliability and is thus an indicator of the correctness of the model derived from this measure. The value of the criterion "Relevance to Reliability" is given in Table 4 for the five measures under consideration. The value $p(M_i)$ is obtained by simple normalization over the spectrum of available models. From the values of $p(M_i)$ in Table 4, the values of θ in Table 3 and the equation used to calculate $E(U(\theta_0-\theta))$, we obtain $p(H_0)$= 0.61.

A simple reference to Table 5 shows that for $\theta_0 = 9\text{x}10^{-2}$ the number of test cases has been reduced from > 72 to 20 when no failures are observed during testing.

Table 4. The value of Relevance to Reliability

Measure	Relevance to Reliability
Code defect density	0.85
Test coverage	0.83
Requirements traceability	0.45
Function point analysis	0.00
Bugs per line of code (Gaffney estimate)	0.00

Table 5. The number of tests required for $\theta_0 = 9\text{x}10^{-2}$, C=0.99

θ_0	$P(H_0)$	n_0
0.09	0.01	72
0.09	0.1	47
0.09	0.2	39
0.09	0.5	25
0.09	0.61	20

6. Summary

We presented two different statistical frameworks for quantification of software reliability based on input domain modeling. The quantification of reliability is obtained either through the sampling model and Bayesian hypothesis testing. The Bayesian framework allows the inclusion of "qualitative" verification and validation activities performed during system's development in terms of prior failure probabilities. These approaches are suitable for our "reliability corroboration" paradigm. To the best of our knowledge this is the first occasion that Bayesian hypothesis testing is proposed for the modeling of software reliability.

The significance of the Bayesian hypothesis testing framework for software reliability corroboration is in the reasonable number of tests that it prescribes for software certification. This is especially obvious in the case of high assurance systems, which for all practical purposes, are considered impossible to certify by today's standards. While this methodology per se does not make systems more reliable than they already are, it provides a framework for quantification of otherwise qualitative software certification processes.

It comes as no surprise that programs developed in stable and mature development environments, which support measurement and process improvement feedback throughout the lifecycle, will require a fewer number of tests for certification. Ultimately, only such environments should be used for high assurance system development. But, as Table 1 indicates, a failure encountered in certification testing imposes a steep economic penalty, because tens of thousands of additional tests may become required.

Regardless of its elegance, the Bayesian hypothesis testing framework for software reliability corroboration may make the practice of software

certification risky if applied inappropriately. The basic current problem is the immaturity of methods for evaluating the precision and trust assigned to pre-certification software reliability beliefs. We have presented an approach to the estimation of prior beliefs based on the RPS theory. This work needs to be pursued. Indeed, the theory should be extended to other software engineering measures, other life-cycle phases. Finally, we need to consider the impact of uncertainties in the measurements and the parameters of the models.

Acknowledgements

This work was supported in part by the NSF CAREER award to the first author and in part by NASA, through cooperative agreement #NCC 2-979 and NRC through Contract #DR-01-0198.

References

[1] "PACS Requirements Specification," Lockheed Martin Corporation Inc., Gaithersburg, MD July 20 1998.

[2] G. Apostolakis, "The Concept of Probability in Safety Assessments of Technological Systems," Science, vol. 250, no.4986 pp. 1359-64, 1990.

[3] F. B. Bastani and A. Pasquini, "Assessment of a Sampling Method for Measuring Safety-Critical Software Reliability," presented at The 3rd International Symposium on Software Reliability Engineering, Monterey, CA, 1994.

[4] R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," IEEE Transactions on Software Engineering, vol. 19, no.1 pp. 3-12, 1993.

[5] G. Cochran, Sampling Techniques. New York: John Wiley & Sons, 1977.

[6] B. Cukic and D. Chakravarthy, "Bayesian Framework for Reliability Assurance of a Deployed Safety-Critical System," presented at The 5th International Symposium on High Assurance Systems (HASE 2000), Albuquerque, NM, 2000.

[7] M. S. Deutsch, Software Verificaiton and Validation: Realistic Project Approaches. Englewood Cliffs, NJ: Prentice Hall, 1982.

[8] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," IEEE Transactions on Software Engineering, vol. 25, no.5 pp. 675-89, 1999.

[9] C. Jones, "The Pragmatics of Software Process Improvements," Software Engineering Technical Council Newsletter, vol. 1, no.3 pp. 269-75, 1996.

[10] M. Li, "On the Nature of Relationships Between Measures and Reliability", Materials and Nuclear Engineering, University of Maryland, College Park, 2002

[11] M. Li and C. Smidts, "Ranking Software Engineering Measures Related to Reliability Using Expert Opinion," presented at The 11th International Symposium on Software Reliability Engineering, San Jose, California, 2000.

[12] A. Mosleh, N. Siu, C. Smidts, and C. Lui, "Model Uncertainty: Its Characterization and Quantification," in International Workshop Series on Advanced Topics in Reliability and Risk Analysis. Annapolis, Maryland: Center for Reliability Engineering, University of Maryland, 1995.

[13] D. L. Parnas, A. J. Van Schouwen, and S. P. Kwan, "Evaluation of Safety-Critical Software," Communications of the ACM, vol. 33, no.6 pp. 636-48, 1990.

[14] C. Smidts and M. Li, "Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems," University of Maryland, Washington D.C. NUREG/GR-0019, November 2000.

[15] C. Smidts, M. Stutzke, and R. W. Stoddard, "Software Reliability Modeling: An Approach to Early Reliability Prediction," IEEE Transactions on Reliability, vol. 47, no.3 pp. 268-78, 1998.

[16] T. A. Thayer, M. Lipow, and E. C. Nelson, Software Reliability. Amsterdam: North-Holland Publishing, 1978.

[17] J. M. Voas, C. C. Michael, and K. W. Miller, "Confidentially Assessing a Zero Probability of Softare Failure," High Integrity Systems, vol. 1, no.3 pp. 269-75, 1995.